

Progetto Sistemi Intelligenti Avanzati

Andrea Barbagallo

A.A. 2019-20

Indice

1	Introduzione	1
2	Algoritmo	1
2.1	Codifica	1
2.2	Funzione di fitness	2
2.3	Popolazione iniziale	2
2.4	Selezione	3
2.5	Ricombinazione	3
2.6	Mutazione	4
3	Interfaccia grafica	6
3.1	Ricerca	6
3.2	Visualizzazione e modifica	7
3.3	Generazione del risultato	8
4	Esempio di utilizzo	9
5	Conclusioni	14
5.1	Risultati finali	14
5.1.1	Tempo di esecuzione	14
5.2	Limiti e problemi	14
5.2.1	Limiti computazionali	15
5.2.2	Limiti derivanti delle librerie	15
5.2.3	Limiti visualizzazione Google Maps	15

1 Introduzione

Lo scopo di questo progetto è quello di sviluppare un'applicazione in grado di risolvere il problema del commesso viaggiatore attraverso l'utilizzo di un algoritmo genetico.

Più precisamente il software dovrà chiedere in input all'utente, una lista di città (o indirizzi) che egli desidera visitare. Successivamente, dovrà essere possibile per l'utente impostare la località di partenza e calcolare, tramite il software, l'itinerario più breve da seguire.

Una volta calcolato, l'itinerario dovrà essere visualizzato sotto forma di lista ordinata, il cui ordine corrisponderà all'ordine in cui visitare le località inserite. Per avere un output più chiaro, sarà inoltre fornito un link tramite il quale visualizzare l'itinerario su Google Maps, attraverso il proprio browser.

2 Algoritmo

Vediamo ora nel dettaglio gli algoritmi e le strutture dati su cui il software si basa per risolvere il problema del commesso viaggiatore.

2.1 Codifica

Per prima cosa occorre definire come rappresentare una possibile soluzione dell'algoritmo, ovvero un percorso che attraversi tutti i luoghi indicati in input dall'utente. Ogni soluzione (e quindi ogni cromosoma) dovrà essere codificato attraverso un vettore ordinato di luoghi, il cui ordine rappresenta come andranno visitati i suddetti luoghi. In questo particolare caso, l'ordine è fondamentale, in quanto da questo dipenderà il valore di fitness associato alla soluzione.

Roma	Milano	Firenze	Venezia	Napoli	...
------	--------	---------	---------	--------	-----

Essendo ogni cromosoma un vettore di luoghi, è necessario ora definire quali informazioni dovrà contenere ogni luogo (e quindi ogni gene).

In questa implementazione, ogni singolo luogo è codificato attraverso tre informazioni:

- Nome
- Latitudine
- Longitudine

Nota Nel software finale la latitudine e la longitudine sono acquisite automaticamente via Internet, in questo modo l'utente dovrà solamente preoccuparsi di inserire il nome del luogo da visitare.

2.2 Funzione di fitness

Dato che l'obiettivo è quello di trovare il percorso più breve possibile, il fitness dovrà necessariamente dipendere dalla lunghezza totale del tragitto.

Supponiamo ora che C sia vettore cromosoma, $C[i]$ il gene in posizione i , e $d(x, y)$ la funzione distanza tra due luoghi x, y . La lunghezza totale D_{tot} del percorso può essere calcolata attraverso la formula

$$D_{tot} = \sum_{i=0}^N d(C[i], C[(i+1) \bmod N])$$

dove N è la lunghezza del cromosoma C (ovvero il numero di luoghi da visitare).

Una volta ottenuta la lunghezza del tragitto, la funzione di fitness può essere definita come $\frac{1}{D_{tot}}$, ovvero l'inverso della lunghezza totale del tragitto. Così facendo, più breve sarà il percorso e più alto sarà il valore di fitness ottenuto.

2.3 Popolazione iniziale

La popolazione iniziale dell'algoritmo è generata in modo casuale. Presa in input la lista di luoghi da visitare, i cromosomi iniziali vengono generati disponendo i luoghi in ordine casuale. Essendo però questo procedimento basato sul caso, è possibile che più cromosomi risultino uguali, e questo potrebbe causare dei problemi.

Per evitare ciò, nel caso in cui più cromosomi generati siano uguali, solo uno di questi sarà inserito e farà quindi parte della popolazione iniziale del programma, mentre gli altri saranno scartati.

2.4 Selezione

Nel software, la fase di selezione avviene attraverso la tecnica della “selezione a roulette”, ossia facendo in modo che la probabilità di ogni cromosoma di essere scelto sia direttamente proporzionale al suo fitness.

Inoltre, in aggiunta a questa selezione, il programma fa affidamento sulla tecnica dell’elitismo, conservando in ogni caso i cromosomi migliori dell’attuale generazione. Questo fa sì che le soluzioni migliori della scorsa generazione siano comunque presenti anche nella successiva.

2.5 Ricombinazione

Dato che nel problema del commesso viaggiatore l’ordine dei geni è di fondamentale importanza per determinare il valore di fitness di un cromosoma, la scelta più saggia è quella di utilizzare un algoritmo che permetta di mantenere questo ordine il più possibile.

Per questo motivo, per eseguire la fase di ricombinazione, il software utilizza un operatore chiamato Partially Mapped Crossover (**PMX**), il cui funzionamento è il seguente:

Supponiamo di avere i seguenti genitori P_1 e P_2 :

$$P_1 = (3 \ 4 \ 8 \ 2 \ 7 \ 1 \ 6 \ 5)$$

$$P_2 = (4 \ 2 \ 5 \ 1 \ 6 \ 8 \ 3 \ 7)$$

Per prima cosa vengono scelti casualmente due punti in cui dividere i genitori, dividendoli così in tre parti ciascuno. In questo caso supponiamo che i punti di divisione siano, uno tra il terzo ed il quarto gene, e l’altro tra il sesto ed il settimo gene (i punti di divisione sono segnati con “|”):

$$P_1 = (3 \ 4 \ 8 \ | \ 2 \ 7 \ 1 \ | \ 6 \ 5)$$

$$P_2 = (4 \ 2 \ 5 \ | \ 1 \ 6 \ 8 \ | \ 3 \ 7)$$

I valori compresi tra i due “|” vengono quindi mappati, in questo esempio avremo: $2 \longleftrightarrow 1$, $7 \longleftrightarrow 6$ e $1 \longleftrightarrow 8$. Ora questi valori vengono scambiati tra i due genitori, ottenendo:

$$O_1 = (\times \times \times \mid 1 \ 6 \ 8 \mid \times \times)$$

$$O_2 = (\times \times \times \mid 2 \ 7 \ 1 \mid \times \times)$$

A questo punto è possibile reinserire i geni (nella stessa posizione in cui erano nel padre), purché questi non generino conflitti, ad esempio non è più possibile inserire 8 nella terza posizione di O_1 in quanto ora è già presente in sesta posizione. Il risultato sarà quindi:

$$O_1 = (3 \ 4 \times \mid 1 \ 6 \ 8 \mid \times \ 5)$$

$$O_2 = (4 \times \ 5 \mid 2 \ 7 \ 1 \mid 3 \times)$$

Infine, per completare O_1 e O_2 si utilizza il mapping effettuato inizialmente. La prima \times di O_1 dovrebbe essere 8, ma essendo questo già presente si controlla il mapping $1 \longleftrightarrow 8$ e vediamo però che anche 1 è già presente. Dato che anche 1 è già presente si controlla nuovamente il mapping $2 \longleftrightarrow 1$, e quindi 2 sarà il valore della prima \times . In modo analogo risolviamo anche la seconda \times di O_1 (che dovrebbe essere 6), usando il mapping $7 \longleftrightarrow 6$ abbiamo che il valore della seconda \times sarà 7:

$$O_1 = (3 \ 4 \ 2 \mid 1 \ 6 \ 8 \mid 7 \ 5)$$

Analogamente viene completato anche O_2 :

$$O_2 = (4 \ 8 \ 5 \mid 2 \ 7 \ 1 \mid 3 \ 6)$$

2.6 Mutazione

Dipendentemente da una certa probabilità, alcuni dei cromosomi risultanti dalla ricombinazione possono subire delle mutazioni. Ogni cromosoma avrà quindi una probabilità di subire una mutazione. Questa mutazione consiste nello scambiare i geni presenti in un certo intervallo.

Per prima cosa vengono estratte casualmente due posizioni del cromosoma, supponiamo che il nostro cromosoma di partenza C sia:

$$C = (5 \ 7 \ 6 \ 1 \ 4 \ 2 \ 8 \ 3)$$

E che le posizioni estratte x , y siano la seconda e la settima:

$$C = (5 \ \underline{7} \ 6 \ 1 \ 4 \ 2 \ \underline{8} \ 3)$$

A questo punto i valori saranno scambiati e le due posizioni si avvicineranno entrambe di “un passo”, questo procedimento si fermerà quando $x = y$ oppure $x > y$ (nel caso in cui i valori fossero numero pari). Nell’esempio le computazioni saranno quindi:

$$C = (5 \ \underline{7} \ 6 \ 1 \ 4 \ 2 \ \underline{8} \ 3)$$

$$C = (5 \ \mathbf{8} \ \underline{6} \ 1 \ 4 \ \underline{2} \ \mathbf{7} \ 3)$$

$$C = (5 \ \mathbf{8} \ \mathbf{2} \ \underline{1} \ \underline{4} \ \mathbf{6} \ \mathbf{7} \ 3)$$

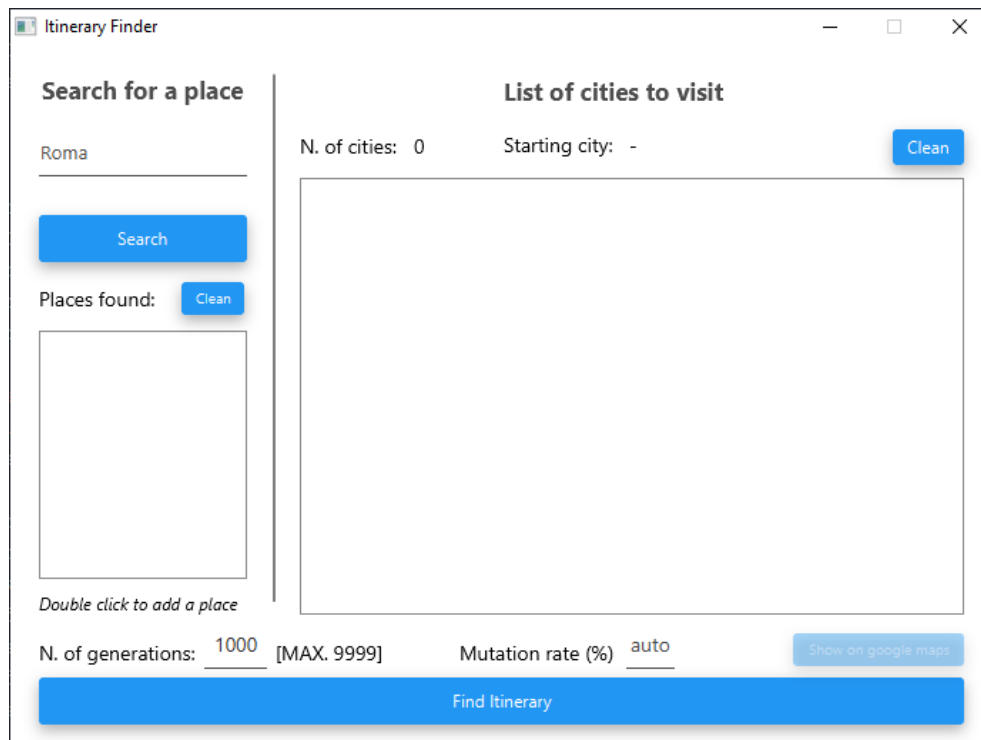
$$C = (5 \ \mathbf{8} \ \mathbf{2} \ \mathbf{4} \ \mathbf{1} \ \mathbf{6} \ \mathbf{7} \ 3)$$

Ed il risultato della mutazione sarà il cromosoma $C = (5 \ 8 \ 2 \ 4 \ 1 \ 6 \ 7 \ 3)$.

Nota La probabilità di mutazione da utilizzare per ottenere un risultato ottimale è difficile da definire, tuttavia utilizzare una probabilità pari a $\frac{1}{N}$ dove N è la dimensione di un cromosoma (numero di gnei) ha dato buoni risultati in fase sperimentale.

3 Interfaccia grafica

Il software è stato sviluppato utilizzando JavaFX e JFoenix, in modo da fornire all'utente un'interfaccia grafica semplice e gradevole. All'avvio, il software presenta la seguente schermata:



È possibile dividere la seguente interfaccia in tre parti principali: a sinistra la parte di **ricerca**, a destra la parte di **visualizzazione e modifica**, ed in basso la parte di **generazione del risultato**.

3.1 Ricerca

Attraverso la barra di ricerca è possibile cercare il luogo che si desidera aggiungere al percorso. La ricerca viene effettuata dal programma utilizzando la libreria *JOpenCage*, che attraverso la rete internet fornisce una lista di luoghi con quel nome. Una volta effettuata la ricerca i risultati trovati appariranno nel blocco *Places found* sotto forma di lista.

Infine, per aggiungere un luogo alla lista dei punti da visitare è sufficiente fare *doppio click* su di esso.

Nota È possibile ripulire questa lista attraverso il pulsante *Clean*, oppure ricercando un'altra località.

3.2 Visualizzazione e modifica

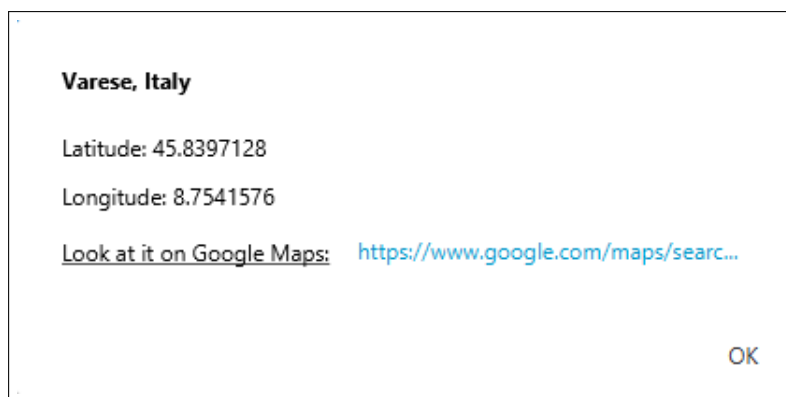
Questa parte contiene tutti i luoghi che si desidera visitare. Inizialmente l'ordine della lista sarà quello di inserimento delle varie città, tuttavia una volta calcolato il risultato (attraverso il pulsante *Find Itinerary*) l'ordine diventerà significativo e rappresenterà l'ordine ottimale di visita dei posti presenti (dall'alto verso il basso).

Inoltre, per ogni luogo presente nella lista l'utente può decidere di:

- Ricevere maggiori informazioni sul luogo.
- Rimuovere il luogo dalla lista.
- Renderlo il luogo di partenza.

Per eseguire una di queste operazioni, è sufficiente fare click con il pulsante destro del mouse su un luogo, e selezionare un'opzione tra “*Info*”, “*Delete*” e “*Make ... the starting city*”.

L'opzione “*Info*” permette di ottenere maggiori informazioni sul luogo, fornendo le sue coordinate geografiche, ed un link che ne permette la visualizzazione su Google Maps.



L'opzione “*Make ... the starting city*” permette di rendere il luogo selezionato, quello di partenza. L'attuale luogo di partenza è reperibile in qualsiasi momento, annotato al di sopra della lista. Infine, l'opzione “*Delete*” rimuove semplicemente il luogo dalla lista.

3.3 Generazione del risultato

Una volta creata la nostra lista di luoghi da visitare, è possibile avviare il processo di generazione della soluzione ottimale. È possibile agire su due parametri:

- Numero di generazioni
- Rateo di mutazione

Da questi parametri dipenderà il tempo di esecuzione necessario, e la qualità della soluzione trovata. Ad esempio aumentando il numero di generazioni il tempo di esecuzione aumenterà, ma la soluzione sarà generalmente più affidabile. Al contrario, riducendo il numero di generazioni, il tempo di esecuzione diminuirà drasticamente, pagando però il prezzo di una soluzione potenzialmente non affidabile.

Una volta avviata la ricerca dell'itinerario migliore, verrà visualizzata una schermata di caricamento, e al termine di questa la lista di verrà riordinata, in base al percorso trovato. La lista rappresenta l'itinerario da seguire (dall'alto verso il basso) per percorrere meno strada possibile.

Se i luoghi non sono più di 10, è possibile visualizzare l'itinerario comodamente su Google Maps, in modo da avere una rappresentazione visiva di come il percorso appare su una cartina geografica. Per visualizzare su Google Maps l'itinerario corrente, fare click su *Show on google maps*.

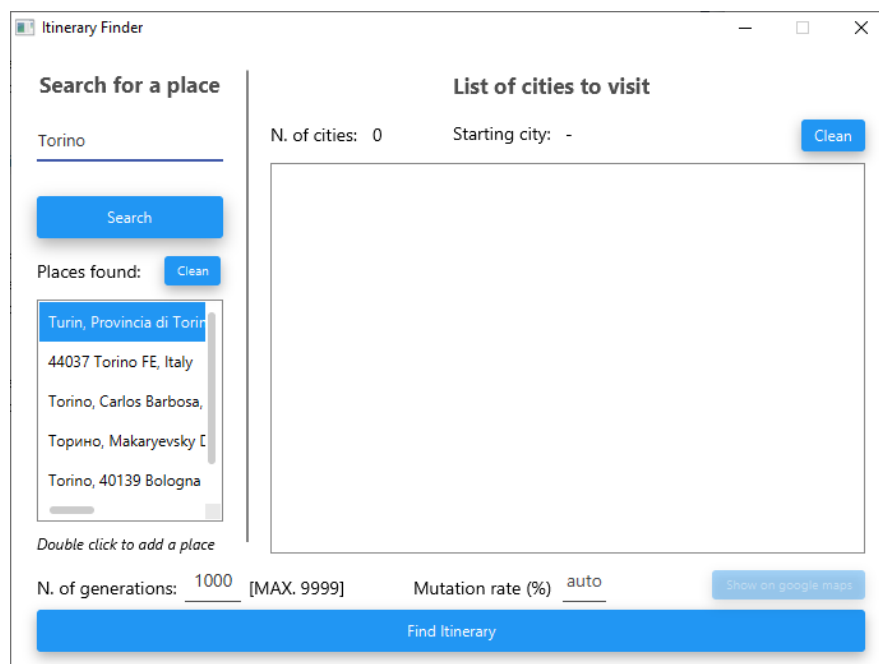
Nota Il numero di generazioni inserito deve essere necessariamente un intero di massimo quattro cifre (massimo 9999), mentre il rateo di mutazione è semplicemente un numero che va da 0 a 100, e rappresenta la frequenza (in percentuale) con cui si avrà una mutazione. Di default il numero di generazioni è impostato a 1000, mentre il mutation rate *automatico* equivale a $\frac{1}{N} \cdot 100$ dove N è la dimensione dei cromosomi.

4 Esempio di utilizzo

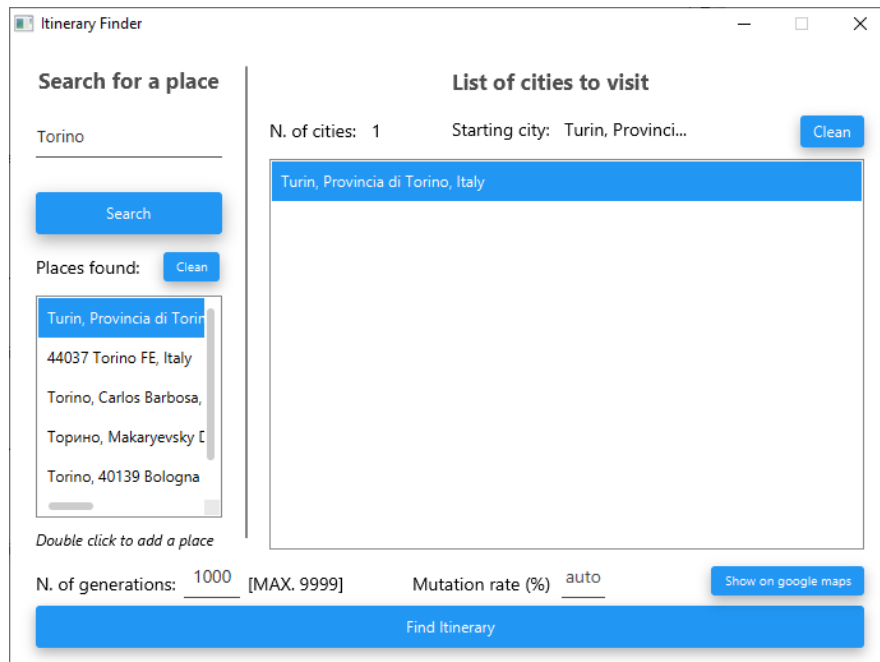
Analizziamo ora le funzionalità fornite dal software, attraverso un esempio di utilizzo. Supponiamo di voler trovare l'itinerario migliore che, partendo da Varese, permetta di visitare le seguenti città italiane: Torino, Cuneo, Alba, Asti, Savona, Genova, Novara, Alessandria, Pavia.

La prima cosa da fare è aggiungere queste città al programma. Per farlo è sufficiente utilizzare la barra di ricerca situata in alto a sinistra per ricercare le varie città da aggiungere.

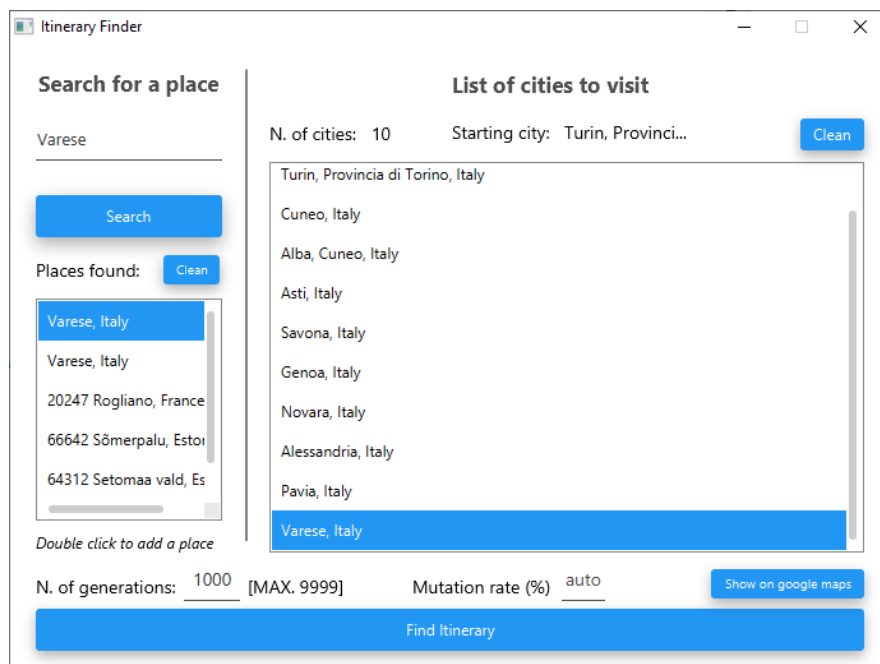
Ad esempio, per aggiungere la città di Torino basterà scriverlo, premere invio (oppure cliccare su “Search”), e al termine della ricerca selezionarlo dai possibili risultati.



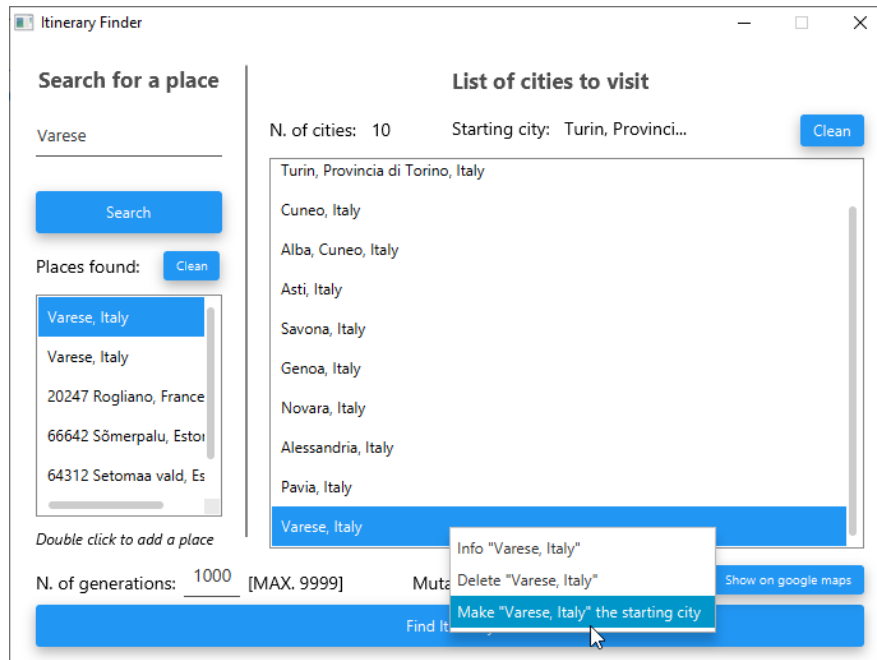
Una volta trovata la città, è sufficiente fare doppio click per aggiungerla alla lista di posti da visitare.



Applicando lo stesso procedimento per tutte le rimanenti città otteniamo:



A questo punto impostiamo “Varese” come città di partenza, in modo che il tragitto calcolato parta e termini nella città di Varese.

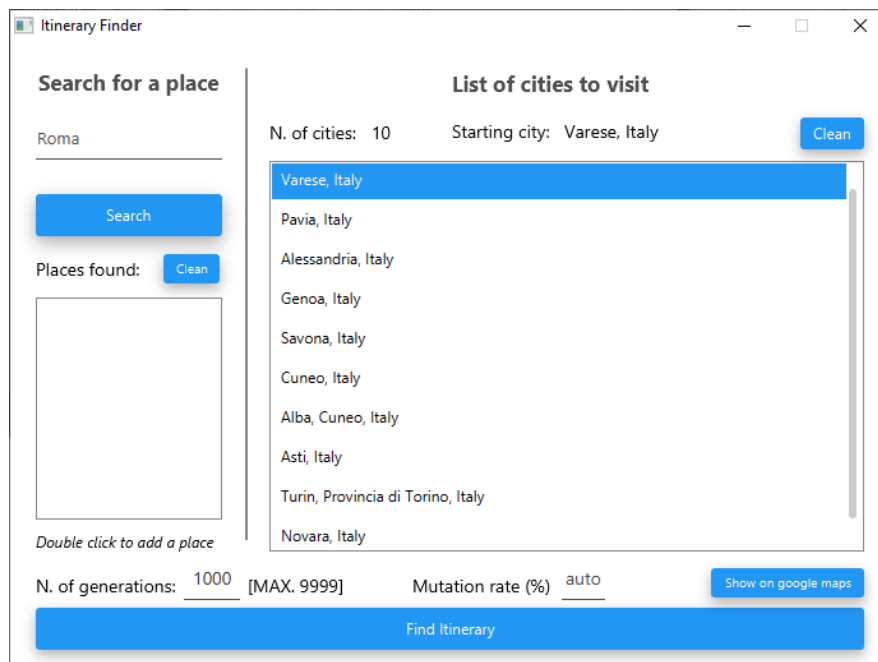


Una volta impostata, dovrebbe apparire in alto a destra, nel formato *Starting city: Varese, Italy*

Infine, utilizziamo semplicemente il pulsante *Find Itinerary* (è possibile modificare prima i parametri se lo si desidera) per trovare l'itinerario migliore (più breve). In questo caso otteniamo l'itinerario:

Varese → Pavia → Alessandria → Genova → Savona → Cuneo → Alba → Asti → Torino → Novara → Varese

come è possibile osservare dall'immagine successiva.

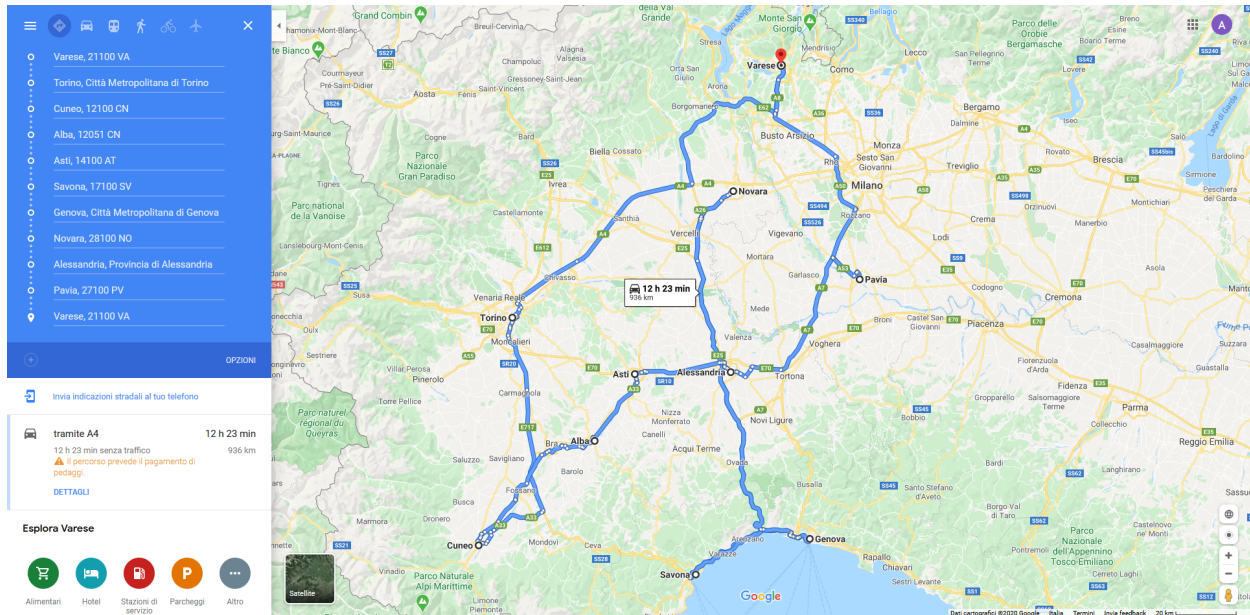


Tramite il pulsante *Show on google maps* è possibile visualizzare la soluzione ottenuta, direttamente su Google Maps.

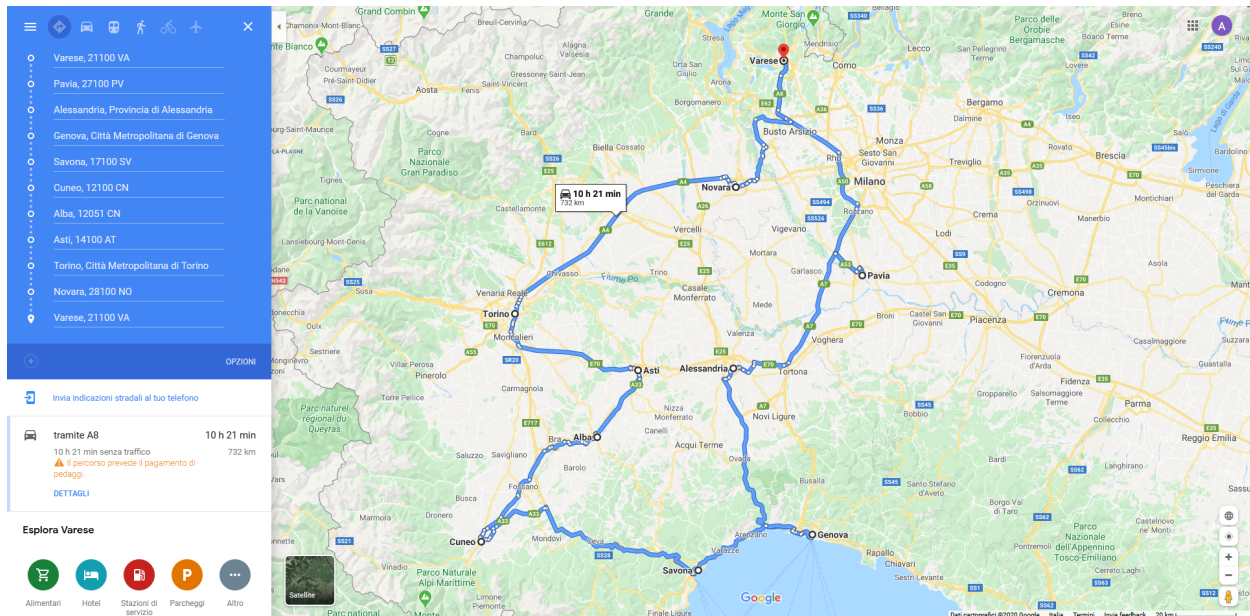
Nel nostro caso, il percorso iniziale aveva una lunghezza di 936 km ed il tempo di percorrenza stimato era di 12 ore e 23 minuti, mentre quello ottenuto al termine di 1000 generazioni (con Mutation rate automatico) ha una lunghezza di 732 km ed un tempo di percorrenza di 10 ore e 21 minuti.

Nella pagina successiva è possibile visualizzare le immagini del percorso prima e dopo l'esecuzione del programma.

Percorso prima dell'esecuzione



Percorso dopo l'esecuzione



5 Conclusioni

5.1 Risultati finali

In media, i risultati ottenuti dal programma riducono la lunghezza del tragitto di circa il **23%**, rispetto ad uno inserito in modo casuale (supponendo di aver scelto un numero di generazioni sufficiente ed un mutation rate adatto).

Nota Il programma è stato testato con insiemi che arrivano fino a 20 città differenti.

5.1.1 Tempo di esecuzione

All'aumentare del numero di generazioni, la complessità dell'algoritmo aumenta, e con essa anche il tempo di esecuzione necessario. Nella tabella seguente sono riportati i tempi di esecuzione (in secondi) misurati al variare del numero di generazioni.

-	Set of 10 cities	Set of 15 cities
N of generations	Execution time (s)	Execution time (s)
500	03,61	04,96
1000	06,94	09,38
2000	14,20	18,46
3000	21,02	29,42
5000	35,68	49,20
9000	64,38	86,23

Come è possibile osservare dai dati, l'aumento del numero di generazioni causa una aumento proporzionale del tempo di esecuzione. Inoltre, è possibile notare come passando da 10 a 15 città, il tempo aumenti solo del 34%, un risultato soddisfacente se si pensa alla complessità computazionale del problema.

5.2 Limiti e problemi

Infine, trovo sia giusto concludere analizzando quelli che sono i limiti e i problemi del software sviluppato.

5.2.1 Limiti computazionali

Data la grande mole di computazioni necessarie per risolvere il problema, aumentando il numero di città da visitare sopra le 20-25 unità, la soluzione ottimale risulterà sempre più difficile da trovare, ed il numero massimo di generazioni (attualmente 9999) potrebbe addirittura diventare insufficiente per trovare una soluzione accettabile.

5.2.2 Limiti derivanti delle librerie

Il software utilizza delle librerie di geo-localizzazione per fornire all'utente la possibilità di cercare semplicemente la città che desidera visitare, evitando di dover inserire manualmente latitudine e longitudine. Questo significa che la lista di città consigliate dopo una ricerca dipende unicamente dalla libreria *JOpenCage*, e di conseguenza non è possibile ottimizzarne il risultato (in alcuni casi la città è difficile da trovare, ed è necessario specificarne la nazione).

5.2.3 Limiti visualizzazione Google Maps

Per la visualizzazione del risultato, il software (oltre a mostrare l'elenco ordinato delle città) permette la visualizzazione su Google Maps. Questa è estremamente comoda, e permette all'utente di avere un risultato più tangibile. Tuttavia, Google permette la visualizzazione di un percorso con al **massimo 10 tappe intermedie**, e per questo motivo non è possibile usufruire di questo servizio per percorsi superiori alle 10 città. E' comunque possibile ricercare manualmente il percorso utilizzando più schede del browser, ma questo è purtroppo un procedimento particolarmente tedioso.